Neural Generalized Predictive Control for Industrial Processes

Sadhana Chidrawar[1], Balasaheb Patre[2] and LaxmanWaghmare[3]
[1]Assistant Professor, MGM's College of Engineering, Nanded (MS) 431 602,
[2,3] Professor, SGGS Institute of Engineering and Technology, Nanded (MS) 431 606 India

1. Introduction

In the manufacturing industry, the requirement for high-speed, fast-response and high-precision performances is critical . Model predictive control (MPC) which, was developed in the late 1970's, refers to a class of computer control algorithms that utilizes an explicit process model to predict the future response of the plant (Qin & Badgwell, 2004). In the last two decades, MPC has been widely accepted for set point tracking and overcoming model mismatch in the refining, petrochemical, chemical, pulp and paper making and food processing industries (Rossiter, 2006). The model predictive control is also introduced to the positioning control of ultra-precision stage driven by a linear actuator (Hashimoto,Goko,et al.,2008). Some of the most popular MPC algorithms that found wide acceptance in industry are Dynamic Matrix Control (DMC), Model Algorithmic Control (MAC), Predictive Functional Control (PFC), Extended Prediction Self Adaptive Control (EPSAC), Extended Horizon Adaptive Control (EHAC) and Generalized Predictive Control (GPC) (Sorensen, Norgaard ,et al., 1999). In most of the controllers, the disturbances arising from manipulated variable are taken care off only after they have already influenced the process output. Thus, there is a necessity to develop the controller to predict and optimize process performance. In MPC the control algorithm that uses an optimizer to solve for the control trajectory over a future time horizon based on a dynamic model of the processes, has become a standard control technique in the process industries over the past few decades. In most applications of model predictive techniques, a linear model is used to predict the process behavior over the horizon of interest. But as most real processes show a nonlinear behavior, some work has to be done to extend predictive control techniques to incorporate nonlinearities of the plant. The most expensive part of the realization of a nonlinear predictive control scheme is the derivation of the mathematical model. In many cases it is even impossible to obtain a suitable physically founded process model due to the complexity of the underlying process or lack of knowledge of critical parameters of the model. The promising way to overcome these problems is to use neural network as a nonlinear models that can approximate the dynamic behavior of the process efficiently

Generalized Predictive Control (GPC) is an independently developed branch of class of digital control methods known as Model Predictive Control (MPC). (Clarke, Mohtadi, et al., 1987) and has become one of the most popular MPC methods both in industry and

academia. It has been successfully implemented in many industrial applications, showing good performance and a certain degree of robustness. It can handle many different control problems for a wide range of plants with a reasonable number of design variables, which have to be specified by the user depending upon a prior knowledge of the plant and control objectives. GPC is known to control non-minimum phase plants, open loop unstable plants and plants with variable or unknown dead time. GPC is robust with respect to modeling errors and sensor noise. The ability of GPC for controlling nonlinear plants and to make accurate prediction can be enhanced if neural network is used to learn the dynamics of the plant. In this Chapter, we have discussed the neural network to form a control strategy known as Neural Generalized Predictive Control (NGPC) (Rao, Murthy, et al., 2006). The NGPC algorithm operates in two modes, i.e. prediction and control. It generates a sequence of future control signals within each sampling interval to optimize control effort of the controlled systems. In NGPC the control vector calculations are made at each sampling instants and are dependent on control and prediction horizon. A computational comparison between GPC and NGPC schemes is given in (Rao, Murthy, et al., 2007).The effect of smaller output horizon in neural generalized predictive control is dealt in (Pitche, Sayyer-Rodsari,et al.,2000). The nonlinear model predictive control using neural network is also developed in (Chen,Yuan,et al.,2002). Two model predictive control (MPC) approaches, an on-line and an off-line MPC approach, for constrained uncertain continuous-time systems with piecewise constant control input are presented (Raff & Sinz, 2008)
Numerous journal articles and meeting papers have appeared on the use of neural network models as the basis for MPC with finite prediction horizons. Most of the publications concentrate on the issues related to constructing neural network models. Very little attention is given to issues of stability or closed-loop performance, although these are still open and unresolved issues. A predictive control strategy based on improved back propagation neural network in order to compensate real time control in nonlinear system with time delays is proposed in (Sun,Chang,et al.,2002).For nonlinear processes, the predictive control would be unsatisfactory. Like neural networks, fuzzy logic also attracted considerable attentions to control nonlinear processes. There are many advantages to control nonlinear system since they has an approximation ability using nonlinear mappings. Generally, they do not use the parametric models such as the form of transfer functions or state space equations. Therefore, the result of modeling or controlling nonlinear systems is not the analytic consequence and we only know that the performance of those is satisfactory. Especially, if the controller requires the parametric form of the nonlinear system, there doesn't exist any ways linking the controller and fuzzy modeling method. The fuzzy model based prediction is derived with output operating point and optimized control is calculated through the fuzzy prediction model using the optimization techniques in (Kim, Ansung, et al.,1998).
In this Chapter, a novel algorithm called Generalized Predictive Control (GPC) is shown to be particularly effective for the control of industrial processes. The capability of the algorithm is tested on variety of systems. An efficient implementation of GPC using a multi-layer feed-forward neural network as the plant's nonlinear model is presented to extend the capability of GPC i.e. NGPC for controlling linear as well as nonlinear process very efficiently. A neural model of the plant is used in the conventional GPC stating it as a neural generalized predictive control (NGPC). As a relatively well-known example, we consider Duffing's nonlinear equation for testing capability of both GPC and NGPC algorithms. The

output of trained neural network is used as the predicted output of the plant. This predicted output is used in the cost function minimization algorithm. GPC criterion is minimized using two different schemes: a Quasi Newton algorithm and Levenberg Marquardt algorithm. GPC and NGPC are applied to the linear and nonlinear systems to test its capability. The performance comparison of these configurations has been given in terms of Integral square error (ISE) and Integral absolute error (IAE). For each system only few more steps in set point were required for GPC than NGPC to settle down the output, but more importantly there is no sign of instability. Performance of NGPC is also tested on a highly nonlinear process of continues stirred tank reactor (CSTR) and linear process dc motor.
The ideas appearing in greater or lesser degree in all the predictive control family are basically:

- Explicit use of a model to predict the process output at future time instants (horizon).
- Calculation of a control sequence minimizing an objective function.
- Receding strategy, so that at each instant the horizon is displaced towards the future, which involves the application of the first control signal of the sequence calculated at each step.

2. MPC Strategy

The methodology of all the controllers belonging to the MPC family is characterized by the following strategy, as represented in Fig.1:

1. The future outputs for a determined horizon *N*, called the prediction horizon, are predicted at each instant *k* using the process model. These predicted outputs $y(t+j/t)$ for $j = 1 \ldots N$ depend on the known values up to instant *t* (past inputs and outputs) and on the future control signals $u(t+j/t)$, $j = 0 \ldots N$-1, which are those to be sent to the system and to be calculated.
2. The set of future control signals is calculated by optimizing a determined criterion in order to keep the process as close as possible to the reference trajectory $w(t+j)$ (which can be the set point itself or a close). This criterion usually takes the form of a quadratic function of the errors between the predicted output signal and the predicted reference trajectory. The control effort is included in the objective function in most cases. An explicit solution can be obtained if the criterion is quadratic, the model is linear and there are no constraints; otherwise an iterative optimization method has to be used. Some assumptions about the structure of the future control law are made in some cases, such as that it will be constant from a given instant.
3. The control signal $u(t/t)$ is sent to the process whilst the next control signal calculated are rejected, because at the next sampling instant $y(t+1)$ is already known and step1 is repeated with this new value and all the sequences are brought up to date. Thus the $u(t+1|t)$ is calculated (which in principle will be different to the $u(t+1|t)$ because of the new information available) using receding horizon control.

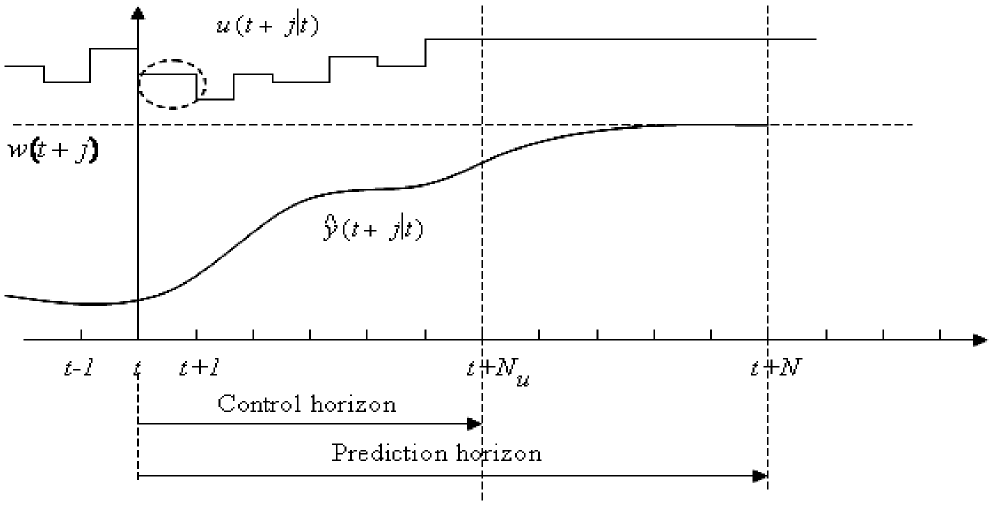


Fig. 1. MPC Strategy

3. Generalized Predictive Controller (GPC)

3.1 Introduction

The basic idea of GPC is to calculate a sequence of future control signals in such a way that it minimizes a multistage cost function defined over a prediction horizon. The index to be optimized is the expectation of a quadratic function measuring the distance between the predicted system output and some reference sequence over the horizon plus a quadratic function measuring the control effort. Generalized Predictive Control has many ideas in common with the other predictive controllers since it is based upon the same concepts but it also has some differences. As will be seen later, it provides an analytical solution (in the absence of constraints), it can deal with unstable and non-minimum phase plants and incorporates the concept of control horizon as well as the consideration of weighting of control increments in the cost function. The general set of choices available for GPC leads to a greater variety of control objective compared to other approaches, some of which can be considered as subsets or limiting cases of GPC. The GPC scheme is shown in Fig. 2. It consists of the plant to be controlled, a reference model that specifies the desired performance of the plant, a linear model of the plant, and the Cost Function Minimization (CFM) algorithm that determines the input needed to produce the plant's desired performance. The GPC algorithm consists of the CFM block. The GPC system starts with the input signal, *r(t),* which is presented to the reference model. This model produces a tracking reference signal, *w(t),* that is used as an input to the CFM block. The CFM algorithm produces an output which is used as an input to the plant. Between samples, the CFM algorithm uses this model to calculate the next control input, *u(t+1)*, from predictions of the response from the plant's model. Once the cost function is minimized, this input is passed to the plant.

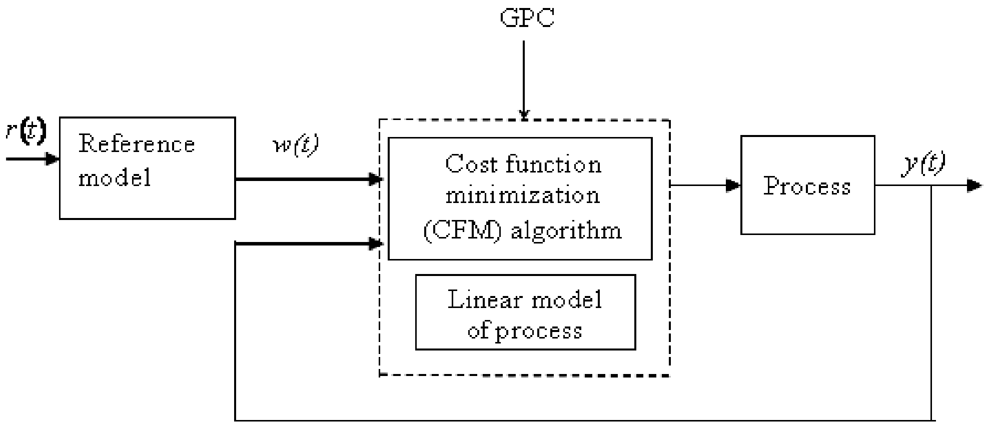


Fig. 2. Basic Structure of GPC

3.2 Formulation of Generalized Predictive Control

Most single-input single-output (SISO) plants, when considering operation around particular set-points and after linearization, can be described by the following:

$$A(z^{-1})y(t) = z^{-d}B(z^{-1})u(t-1) + C(z^{-1})e(t) \tag{1}$$

where $u(t)$ and $y(t)$ are the control and output sequence of the plant and $e(t)$ is a zero mean white noise. A, B and C are the following polynomials in the backward shift operator z^{-1}:

$$A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + \dots\dots + a_{na} z^{-na}$$
$$B(z^{-1}) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots\dots + b_{nb} z^{-nb}$$
$$C(z^{-1}) = 1 + c_1 z^{-1} + c_2 z^{-2} + \dots\dots + c_{nc} z^{-nc}$$

where, d is the dead time of the system. This model is known as a Controller Auto-Regressive Moving-Average (CARMA) model. It has been argued that for many industrial applications in which disturbances are non-stationary an integrated CARMA (CARIMA) model is more appropriate. A CARIMA model is given by,

$$A(z^{-1})y(t) = z^{-d}B(z^{-1})u(t-1) + C(z^{-1})\frac{e(t)}{\Delta} \tag{2}$$

with $\Delta = 1 - z^{-1}$

For simplicity, C polynomial in (2) is chosen to be 1. Notice that if C^{-1} can be truncated it can be absorbed into A and B.

3.3 Cost Function

The GPC algorithm consists of applying a control sequence that minimizes a multistage cost function,

$$J(N_1, N_2, N_u) = \sum_{j=N_1}^{N_2} \delta(j)[\hat{y}(t+j|t) - w(t+j)]^2 + \sum_{j=1}^{N_u} \lambda(j)[\Delta u(t+j-1)]^2 \quad (3)$$

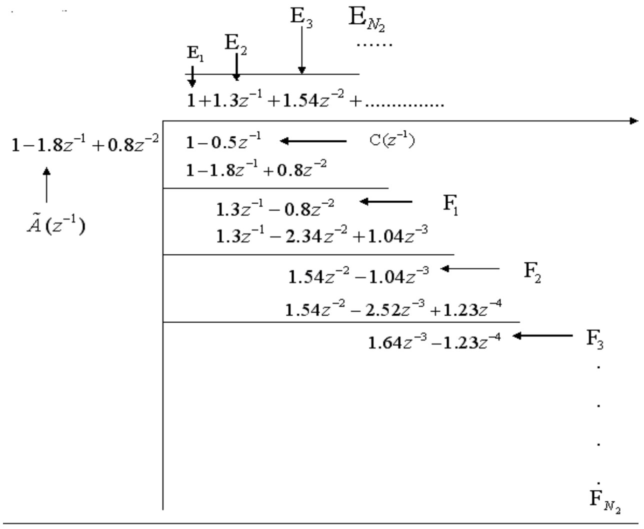
where $\hat{y}(t+j\,|\,t)$ is an optimum *j*-step ahead prediction of the system output on data up to time *k*, N_1 and N_2 are the minimum and maximum costing horizons, N_u control horizon, $\delta(j)$ and $\lambda(j)$ are weighing sequences and $w(t+j)$ is the future reference trajectory, which can considered to be the constant.
The objective of predictive control is to compute the future control sequence $u(t)$, $u(t+1), \ldots u(t+N_u)$ in such a way that the future plant output $y(t+j)$ is driven close to $w(t+j)$. This is accomplished by minimizing $J(N_1, N_2, N_u)$.

3.4 Cost Function Minimization Algorithm

In order to optimize the cost function the optimal prediction of $y(t+j)$ for $j \geq N_1$ and $j \leq N_2$ is required. To compute the predicted output, consider the following Diophantine equation,

$$1 = E_j(z^{-1})\tilde{A}(z^{-1}) + z^{-j}F_j(z^{-1}) \quad \text{with } \tilde{A}(z^{-1}) = \Delta A(z^{-1}) \quad (4)$$

The polynomials E_j and F_j are uniquely defined with degrees $j-1$ and na respectively. They can be obtained dividing 1 by $\tilde{A}(z^{-1})$ until the remainder can be factorized as $z^{-j}F_j(z^{-1})$. The quotient of the division is the polynomial $E_j(z^{-1})$. An example demonstrating calculation of E_j and Fj coefficients in Diophantine equation (4) is shown in Example 1 given below:



Example 1. Diophantine equation demonstration example

If (2) is multiplied by $\Delta E_j(z^{-1})z^j$ we get,

$$\tilde{A}(z^{-1})E_j(z^{-1})y(t+j) = E_j(z^{-1})B(z^{-1})\Delta u(t+j-d-1) + E_j(z^{-1})e(t+j) \tag{5}$$

Substituting (4), in (5) we get,

$$(1 - z^{-j}F_j(z^{-1}))y(t+j) = E_j(z^{-1})B(z^{-1})\Delta u(t+j-d-1)) + F_j(z^{-1})y(t) + E_j(z^{-1})e(t+j)$$

which can be rewritten as:

$$y(t+j) = F_j(z^{-1})y(t) + E_j(z^{-1})B(z^{-1})\Delta u(t+j-d-1)) + F_j(z^{-1})y(t) + E_j(z^{-1})e(t+j) \tag{6}$$

degree of polynomial is $E_j(z^{-1}) = j-1$ the noise terms in equation are all in the future. The best prediction of $y(t+j)$ is given by,

$$\hat{y}(t+j \mid t) = G_j(z^{-1})\Delta u(t+j-d-1) + F_j(z^{-1})y(t) \tag{7}$$

where, $G_j(z^{-1}) = E_j(z^{-1})B(z^{-1})$.

It is very simple to show that, the polynomials E_{j+1} and F_j can be obtained recursively. Consider that polynomials E_j and F_j have been obtained by dividing 1 by $\tilde{A}(z^{-1})$ until the remainder of the division can be factorized as $z^{-j}F_j(z^{-1})$. These polynomials can be expressed as:

$$F_j(z^{-1}) = f_{j,0} + f_{j,1}z^{-1} + f_{j,2}z^{-2} + \dots\dots + f_{j,na}z^{-na} \tag{8}$$

$$E_j(z^{-1}) = e_{j,0} + e_{j,1}z^{-1} + e_{j,2}z^{-2} + \dots\dots + e_{j,j-1}z^{-(j-1)} \tag{9}$$

Suppose that the same procedure is used to obtain E_{j+1} and F_{j+1}, that is, dividing 1 by $\tilde{\mathbf{A}}(z^{-1})$ until the remainder of the division can be factorized as $z^{-(j-1)}F_{j+1}(z^{-1})$ with

$$F_{j+1}(z^{-1}) = f_{j+1,0} + f_{j+1,1}z^{-1} + \dots\dots + f_{j+1,na}z^{-na} \tag{10}$$

It is clear that only another step of the division is performed to obtain the polynomials E_{j+1} and F_{j+1}. The polynomial E_{j+1} will be given by:

$$E_{j+1}(z^{-1}) = E_j(z^{-1}) + e_{j+1,j}z^{-j} \tag{11}$$

with $e_{j+1,j} = f_{j,0}$. The coefficients of polynomial F_{j+1} can then be expressed as:

$$f_{j+1,i} = f_{j+1,i+1} - f_{j,0}\tilde{a}_{i+1} \qquad i = 0 \;....\; na-1 \tag{12}$$

The Polynomial G_{j+1} can be obtained recursively as follows:

$$G_{j+1} = E_{j+1}B = (E_j + f_{j,0}z^{-1})B \tag{13}$$

$$G_{j+1} = G_j + f_{j,0}z^{-1}B \tag{14}$$

That is, the first *j* coefficient of G_{j+1} will be identical to those of G_j and the remaining coefficients will be given by:

$$g_{j+1,j+i} = g_{j,j+i} + f_{j,0}b_i \qquad \text{for } i = 0 \;.....\; nb \tag{15}$$

To solve the GPC problem the set of control signals $u(t)$, $u(t+1)$, …, $u(t+N)$ has to be obtained in order to optimize expression. As the system considered has a dead time d sampling periods, the output of the system will be influenced by signal $u(t)$ after sampling period $d+1$. The values N_1, N_2 and N_u defining the horizon can be defined

by $N_2 = d + N$, $N_2 = d + N$ and $N_u = N$. Notice that there is no point in making $N_1 > d + 1$ as terms added to expression will only depend on the past control signals. On the other hand, if $N_1 > d + 1$, the first point in the reference sequence, being the ones guessed with most certainly, will not be taken into account.
Now consider the following set of *j* ahead optimal predictions given below:

$$\begin{aligned} \hat{y}(t+d+1 \mid t) &= G_{d+1}\Delta u(t) + F_{d+1}y(t) \\ \hat{y}(t+d+2 \mid t) &= G_{d+2}\Delta u(t+1) + F_{d+2}y(t) \\ \hat{y}(t+d+N \mid t) &= G_{d+N}\Delta u(t+N-1) + F_{d+N}y(t) \end{aligned} \tag{16}$$

which can be re-written as:

$$y = \mathbf{G}u + \mathbf{F}(z^{-1})y(t) + \mathbf{G}'(z^{-1})\Delta u(t-1) \tag{17}$$

where,

$$y = \begin{bmatrix} \hat{y}(t+d+1 \mid t) \\ \hat{\hat{y}}(t+d+2 \mid t) \\ \cdots\cdot \\ \hat{y}(t+d+N \mid t) \end{bmatrix}; \; u = \begin{bmatrix} \Delta u(t) \\ \Delta u(t+1) \\ \cdots\cdot \\ \Delta u(t+N-1) \end{bmatrix}; \; \mathbf{G} = \begin{bmatrix} g_0 & 0 & \cdots\cdot & 0 \\ g_1 & g_0 & \cdots\cdot & 0 \\ . & . & . & . \\ g_{N-1} & g_{N-2} & \cdots\cdot & g_0 \end{bmatrix}$$

$$\mathbf{G}'(z^{-1}) = \begin{bmatrix} (\mathbf{G}_{d+1}(z^{-1}) - g_0)z \\ (\mathbf{G}_{d+2}(z^{-1}) - g_0 - g_1 z^{-1})z^2 \\ \cdots\cdots \\ (\mathbf{G}_{d+N}(z^{-1}) - g_0 - g_1 z^{-1} \cdots\cdots g_{N-1} z^{-(N-1)})z^N \end{bmatrix}; \; \mathbf{F}(z^{-1}) = \begin{bmatrix} F_{d+1}(z^{-1}) \\ F_{d+2}(z^{-1}) \\ \cdots\cdot \\ F_{d+N}(z^{-1}) \end{bmatrix}$$

Note that if the plant dead time is $d > 1$ the first d - 1 rows of **G** will be null, but if instead N_1 is assumed to be equal to d the leading element is non-zero. However, as d will not in general be known in the self-tuning case one key feature of the GPC approach is that a stable solution is possible even if the leading rows of **G** are zero.
Notice that the last two terms in (17) only depends on the past and can be grouped into f leading to:

$$\hat{y} = \mathbf{G}\,\mathrm{u} + \mathrm{f}$$

Notice that if all initial conditions are zero, the free response f is also zero. If a unit step is applied to the input at time *t;* that is

$$\Delta u(t) = 1, \Delta u(t+1) = 0, \; \ldots, \; \Delta u(t+N-1) = 0$$

then expected output sequence $[\hat{y}(t+1), \hat{y}(t+2), \ldots\ldots \hat{y}(t+N)]^{\mathrm{T}}$ is equal to the first column of matrix $\mathbf{G}$. That is, the first column of matrix $\mathbf{G}$ can be calculated as the step response of the plant when a unit step is applied to the manipulated variable. The free response term can be calculated recursively by:

$$\mathbf{f}_{j+1} = z(1 - \tilde{A}(z^{-1}))\mathbf{f}_j + B(z^{-1})\Delta u(t-d+j) \tag{18}$$

with $\mathrm{f}_0 = y(t)$ and $\Delta u(t+j) = 0$ for $j \geq 0$.
The equation (3) can be written as:

$$J = (Gu + f - w)^T (Gu + f - w) + \lambda u^T u \tag{19}$$

where,

$$w = [w(t+d+1) \quad w(t+d+2) \quad ..... \quad w(t+d+N)]^T$$

It has been considered that the future reference trajectory keeps constant along the horizon or its evolution is unknown and therefore $w(t+i) = w(t)$.

The equation (19) can be written as:

$$J = \frac{1}{2} u^T H u + b^T u + f_0 \tag{20}$$

where,

$$\mathbf{H} = 2(\mathbf{G}^T \mathbf{G} + \mathrm{R\,I})$$

$$b^T = 2(\mathrm{f\text{-}w})^T \mathbf{G}$$

$$\mathrm{f}_0 = (\mathrm{f\text{-}w})^T (\mathrm{f\text{-}w})$$

The minimum of *J*, assuming there are no constraints on the control signals, can be found by making the gradient of J equal to zero, which leads to:

$$u = -H^{-1} b = (G^T G + \lambda I)^{-1} G^T (w - f) \tag{21}$$

The dimension of the matrix involved in equation (19) is *N* x *N*. Although in the non-adaptive case the inversion need be performed only once. In a self-tuning version the computational load of inverting at each sample would be excessive. Moreover, if the wrong value for dead-time is assumed, $G^T G$ is singular and hence a finite non-zero value of weighting λ would be required for a realizable control law, which is inconvenient because the accurate value for λ would not be known a priori. Notice that the control signal that is actually sent to the process is the first element of vector u , which is given by:

$$\Delta u = K(w - f) \tag{22}$$

where *K* is the first row of matrix $(G^T G + \lambda I)^{-1} G^T$. If there are no future predicted errors, that is, if $(w - f) = 0$, then there is no control move, since the objective will be fulfilled with the free evolution of the process. However, in the other case, there will be an increment in the control action proportional (with a factor *K*) to that future error. Notice that the action is taken with respect to *future* errors, not *past* errors, as is the case in conventional feedback controllers.

Also notice that when only the first element of u is applied, then at the next sampling instants, new data are acquired and a new set of control moves is calculated. Once again, only the first control move is implemented. These activities repeated at each sampling instant, and the strategy is referred to as a *receding horizon approach*. It may strange to calculate an N_u -step control policy and then only implement the first move. The important advantage of this receding horizon approach is that new information in the form of the most recent measurements *y*(*k*) is utilized immediately instead of being ignored for the next N_u sampling instants. Otherwise, the multi-step predictions and control moves would be based on old information and thus be adversely affected by unmeasured disturbances.

4. Introduction to Neural Generalized Predictive Control

The Generalized Predictive Control (GPC), introduced in above section, belongs to a class of digital control methods called Model-Based Predictive Control (MBPC). GPC is known to control a non-minimum phase plants, open-loop unstable plants and plants with variable or unknown dead time. GPC had been originally developed with linear plant predictor models which, leads to a formulation that can be solved analytically. But most of the real processes show nonlinear behavior. Some work has to be done to extend the predictive control techniques to incorporate nonlinear models. Developing adequate nonlinear empirical models is very difficult and there is no model form that is clearly suitable to represent general nonlinear processes. Part of the success of standard model based predictive techniques was due to the relative ease with which step and impulse responses or low order transfer functions could be obtained. A major mathematical obstacle to complete theory of nonlinear processes is the lack of superposition principal for nonlinear systems. The selection of the minimization algorithm affects the computational efficiency of the algorithm. Explicit solution for it can be obtained if the criterion is quadratic, the model is linear and there are no constraints; otherwise an iterative optimization method has to be used. In this work a Newton-Raphson method is used as the optimization algorithm. The main cost of the Newton-Raphson algorithm is in the calculation of the Hessian, but even with this overhead the low iteration numbers make Newton-Raphson a faster algorithm for real-time control (Soloway & Haley,1997).

The Neural Generalized Predictive Control (NGPC) scheme is shown in Fig. 3. It consists of four components, the plant to be controlled, a reference model that specifies the desired performance of the plant, a neural network that models the plant, and the Cost Function Minimization (CFM) algorithm that determines the input needed to produce the plant's desired performance. The NGPC algorithm consists of the CFM block and the neural net block.

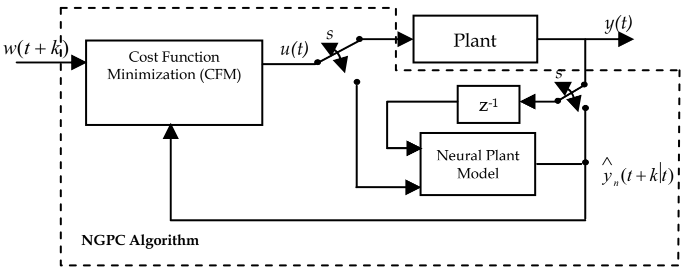


Fig. 3. Block Diagram of NGPC System

The NGPC system starts with the input signal, *w(t),* which is applied to the reference model. This model produces a tracking reference signal, *w(t+k),* that is used as an input to the CFM block. The CFM algorithm produces an output which is either used as an input to the plant or the plant's model. The double pole double throw switch, S, is set to the plant when the CFM algorithm has solved for the best input, *u(n),* that will minimize a specified cost function. Between samples, the switch is set to the plant's model where the CFM algorithm uses this model to calculate the next control input, *u(n+1)*, from predictions of the response from the plant's model. Once the cost function is minimized, this input is passed to the plant. The computational performance of a GPC implementation is largely based on the minimization algorithm chosen for the CFM block. The selection of a minimization method can be based on several criteria such as: number of iterations to a solution, computational costs and accuracy of the solution. In general these approaches are iteration intensive thus making real-time control difficult. In this work Newton-Raphson as an optimization technique is used. Newton-Raphson is a quadratically converging. The improved convergence rate of Newton-Raphson is computationally costly, but is justified by the high convergence rate of Newton-Raphson. The quality of the plant's model affects the accuracy of a prediction. A reasonable model of the plant is required to implement GPC. With a linear plant there are tools and techniques available to make modeling easier, but when the plant is nonlinear this task is more difficult. Currently there are two techniques used to model nonlinear plants. One is to linearize the plant about a set of operating points. If the plant is highly nonlinear the set of operating points can be very large. The second technique involves developing a nonlinear model which depends on making assumptions about the dynamics of the nonlinear plant. If these assumptions are incorrect the accuracy of the model will be reduced. Models using neural networks have been shown to have the capability to capture nonlinear dynamics**.** For nonlinear plants, the ability of the GPC to make accurate predictions can be enhanced if a neural network is used to learn the dynamics of the plant instead of standard modeling techniques. Improved predictions affect rise time, over-shoot, and the energy content of the control signal.

5. Formulation of NGPC

5.1 Cost Function

As mentioned earlier, the NGPC algorithm is based on minimizing a cost function over a finite prediction horizon. The cost function of interest to this application is,

$$J(N_1, N_2, N_u) = \sum_{j=N_1}^{N_2} \delta(j)\left[\hat{y}_n(t+j|t) - w(t+j)\right]^2 + \sum_{j=1}^{N_u} \lambda(j)\left[\Delta u(t+j-1)\right]^2 \quad (23)$$

N_1 = Minimum costing prediction horizon

N_2 = Maximum costing prediction horizon

N_u= Length of control horizon

$\hat{y}(t+k|t)$ = Predicted output from neural network

$u(t+k|t)$ = Manipulated input

$w(t+k)$ = Reference trajectory

δ and λ = Weighing factor

This cost function minimizes not only the mean squared error between the reference signal and the plant's model, but also the weighted squared rate of change of the control input with it's constraints. When this cost function is minimized, a control input that meets the constraints is generated that allows the plant to track the reference trajectory within some tolerance. There are four tuning parameters in the cost function, N_1, N_2, N_u, and λ**.** The predictions of the plant will run from N_1 to N_2 future time steps. The bound on the control horizon is N_u. The only constraint on the values of N_u and N_1 is that these bounds must be less than or equal to N_2. The second summation contains a weighting factor, λ that is introduced to control the balance between the first two summations. The weighting factor acts as a damper on the predicted *u(n+1).*

5.2 Cost Function Minimization Algorithm

The objective of the CFM algorithm is to minimize *J* in equation (24) with respect to *[u(n+l), u(n+2), ..., u(n+*N_u*)]*^{T}, denoted as *U*. This is accomplished by setting the Jacobian of equation (23) to zero and solving for *U*. With Newton-Rhapson used as the CFM algorithm, *J* is minimized iteratively to determine the best *U*. An iterative process yields intermediate values for *J* denoted *J(k)*. For each iteration of *J(k)* an intermediate control input vector is also generated and is denoted as:

$$U(k) \overset{\Delta}{=} \begin{bmatrix} u(t+1) \\ u(t+2) \\ . \\ . \\ . \\ u(t+N_u) \end{bmatrix} \qquad k = 1, \; .....N_u \tag{24}$$

The Newton-Raphson method is one of the most widely used of all root-locating formula. If the initial guess at the root is x_i, a tangent can be extended from the point [x_i, $f(x_i)$]. The point where this tangent crosses the *x* axis usually represents an improved estimate of the root. So the first derivative at *x* on rearranging can be given as: $x_{(i+1)} = x_{(i)} - \dfrac{f(x_i)}{f^{'}(x_i)}$

Using this Newton-Raphson update rule, $U(k+1)$ is given by,

$$U(k+1) = U(k) - \left(\frac{\partial^2 J}{\partial U^2}(k)\right)^{-1} \frac{\partial J}{\partial U}(k)\text{'} \quad \text{where} \quad f(x) = \frac{\partial J}{\partial U} \tag{25}$$

and the Jacobian is denoted as,

$$\frac{\partial J}{\partial U}(k) \equiv \begin{bmatrix} \dfrac{\partial J}{\partial u(t+1)} \\ . \\ . \\ . \\ \dfrac{\partial J}{\partial u(t+N_u)} \end{bmatrix} \tag{26}$$

Also the Hessian is given by,

$$\frac{\partial^2 J}{\partial U^2}(k) \equiv \begin{bmatrix} \frac{\partial^2 J}{\partial u(t+1)^2} & \cdot\ \cdot & \frac{\partial^2 J}{\partial u(t+1)\partial u(t+N_u)} \\ \cdot \quad \cdot & \cdot\ \cdot & \cdot \\ \cdot & \cdot\ \cdot & \cdot \\ \frac{\partial^2 J}{\partial u(t+N_u)\partial u_k(t+1)} & \cdot\ \cdot & \frac{\partial^2 J}{\partial u(t+N_u)^2} \end{bmatrix} \quad (27)$$

Each element of the Jacobian is calculated by partially differentiating equation (23) with respect to vector *U*.

$$\frac{\partial J}{\partial u(t+h)} = 2\sum_{j=N_1}^{N_2} \delta(j)\left[\hat{y}_n(t+j) - w(t+j)\right]\frac{\partial \hat{y}_n(t+j)}{\partial u(t+h)} + 2\sum_{j=1}^{N_u} \lambda(j)\left[\Delta u(t+j)\right]\frac{\partial \Delta u(t+j)}{\partial u(t+h)} \quad (28)$$

where, $h = 1, \ldots, N_u$.

Once again equation (28) is partially differentiated with respect to vector *U* to get each element of the Hessian.

$$\frac{\partial^2 J}{\partial u(t+m)\partial u(t+h)} = 2\sum_{j=N_1}^{N_2} \delta(j)\left\{\frac{\partial^2 \hat{y}(t+j)}{\partial u(t+m)\partial u(t+h)}\left[\hat{y}(t+j) - w(t+j)\right] - \frac{\partial \hat{y}(t+j)}{\partial u(t+m)}\frac{\partial \hat{y}(t+j)}{\partial u(t+h)}\right\}$$

$$+2\sum_{j=N_1}^{N_2} \lambda(j)\left\{\frac{\partial \Delta n(t+j)}{\partial u(t+m)}\frac{\partial \Delta n(t+j)}{\partial u(t+h)} + \frac{\partial^2 \Delta n(t+j)}{\partial u(t+m)\partial u(t+h)}\right\} \quad (29)$$

The m^{th}, h^{th} elements of the Hessian matrix in equation (27) are, $h = 1, \ldots, N_u$ and $m = 1, \ldots, N_u$.

The last computation needed to evaluate $U(k+1)$ is the calculation of the predicted output of the plant, $\hat{y}(t+j)$, and it's derivatives. The next sections define the equation of a multilayer feed forward neural network, and define the derivative equations of the neural network.

6. Neural Network for Prediction

In NGPC the model of the plant is a neural network. This neural model is constructed and trained using MATLAB Neural Network System Identification Toolbox commands (Norgaard, 2000). The output of trained neural network is used as the predicted output of the plant. This predicted output is used in the Cost Function Minimization Algorithm. If *$y_n(t)$* is the neural network's output then it is nothing but plant's predicted output $\hat{y}_n(t+k|t)$. The initial training of the neural network is typically done off-line before control is attempted. The block configuration for training a neural network to model the plant is shown in Fig. 4. The network and the plant receive the same input, *u(t).* The network has an additional input that either comes from the output of the plant, *y(t),* or the neural network's, *$y_n(t)$.* The one that is selected depends on the plant and the application. This input assists the network with capturing the plant's dynamics and stabilization of unstable systems. To train the network, its weights are adjusted such that a set of inputs produces the desired set of outputs. An error is formed between the responses of the network, *$y_n(t)$*, and the plant, *y(t).* This error is then used to update the weights of the network through gradient descent learning. In this work, a Levenberg-Marquardt method is used as gradient descent learning algorithm for updating the weights. This is standard method for minimization of mean-square error criteria, due to its rapid convergence properties and robustness. This process is repeated until the error is reduced to an acceptable level. Since a neural network is used to model the plant, the configuration of the network architecture should be considered. This implementation of NGPC adopts input/output models.

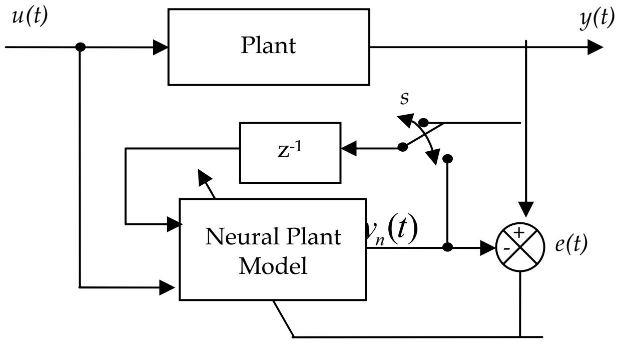


Fig. 4.Block Diagram of Off-line Neural Network Training

The diagram shown in Fig. 5, depicts a multi-layer feed-forward neural network with a time delayed structure. For this example, the inputs to this network consists of two external inputs, *u(t)* and two outputs *y(t-1),* with their corresponding delay nodes, *u(t), u(t-1)* and *y(t-1), y(t-2).* The network has one hidden layer containing five hidden nodes that uses bi-polar sigmoidal activation output function. There is a single output node which uses a linear output function, of one for scaling the output.

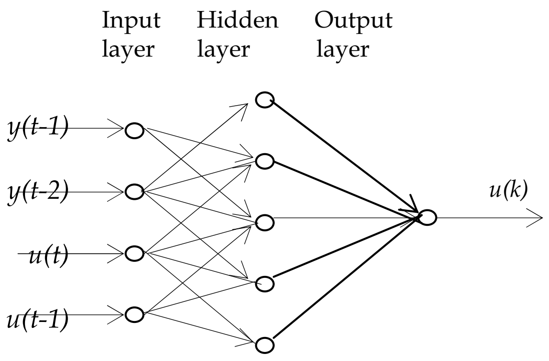


Fig. 5.Neural Network Architecture

The equation describing this network architecture is:

$$y_n(t) = \sum_{j=1}^{hid} w_j f_j\left(net(t)\right) \tag{30}$$

and

$$net_j(t) = \sum_{j=1}^{n_d} w_{j,i+1} u(t-i) + \sum_{j=1}^{d_d} w_{j,n_d+i+1} y(t-i) \tag{31}$$

where,

$y_n(t)$	is the output of the neural network
$f_j(.)$	is the output function for the j^{th} node of the hidden layer
$net_j(t)$	is the activation level of the j^{th} node's output function
hid	is the number of hidden nodes in the hidden layer
n_d	is the number of input nodes associated with *u(.)*
d_d	is the number of input nodes associated with *y(.)*
w_j	is the weight connecting the j^{th} hidden node to the output node
$w_{j,i}$	is the weight connecting the i^{th} hidden input node to the j^{th} hidden node
$y(t-i)$	is the delayed output of the plant used as input to the network
$u(t-i)$	is the input to the network and its delays

This neural network is trained in offline condition with plants input/output data.

Prediction Using Neural Network

The NGPC algorithm uses the output of the plant's model to predict the plant's dynamics to an arbitrary input from the current time, *t*, to some future time, *t*+*k*. This is accomplished by time shifting equations equation (30) and (31), by *k*, resulting in the following equations given by,

$$y_n(t+k) = \sum_{j=1}^{hid} w_j f_j\left(net_j(t+k)\right) \tag{32}$$

and

$$net_j(t+k) = \sum_{i=1}^{hid} w_{j,i+1} \begin{cases} u(n+k-i), k-N_u < i \\ u(n+N_u), k-N_u \geq i \end{cases} + \sum_{j=1}^{\min(k,d_d)} \left(w_{j,n_d+i+1} y_n(t+k-i)\right)$$
$$+ \sum_{i=k+1}^{d_d} \left(w_{j,n_d+i+1} y(t+k-i)\right) \tag{33}$$

The first summation in equation (33) breaks the input into two parts represented by the conditional. The condition where $k - N_u < i$ handles the previous future values of the u up to *u(t+Nu-1)*.The condition where $k - N_u > i$ sets the input from $u(t+N_u)$ to $u(t+k)$ equal to $u(t+N_u)$. The second condition will only occur if $N_2 > N_u$. The next summation of equation (33) handles the recursive part of prediction. This feeds back the network output, y_n, for k or d_d times, which ever is smaller. The last summation of equation (33) handles the previous values of *y*. The following section derives the derivatives of equation (32) and (33) with respect to the input $u\left(t+h\right)$.

6.1 Neural Network Derivative Equations

To evaluate the Jacobian and the Hessian in equation (26) and (27) the network's first and second derivative with respect to the control input vector are needed.

Jacobian Element Calculation:

The elements of the Jacobian are obtained by differentiating *y*$_n$*(t+k)* in equation (32) with respect to *u(t+h)* resulting in

$$\frac{\partial yn\left(t+k\right)}{\partial u\left(t+h\right)} = \sum_{j=1}^{hid} w_j \frac{\partial f_j\left(net_j\left(t+k\right)\right)}{\partial u\left(t+h\right)} \tag{34}$$

Applying chain rule to $\partial f_j\left(net_j\left(t+k\right)\right) / \partial u\left(t+h\right)$ results in

$$\frac{\partial f_j\left(net_j\left(t+k\right)\right)}{\partial u\left(t+h\right)} = \frac{\partial f_j\left(net_j\left(t+k\right)\right)}{\partial net_j\left(t+k\right)} \frac{\partial net_j\left(t+k\right)}{\partial u\left(t+h\right)} \tag{35}$$

where $\partial f_j\left(net_j\left(t+k\right)\right) / \partial net_j\left(t+k\right)$ is the output function's derivative which will become zero as we are using a linear (constant value) output activation function and

$$\frac{\partial net_j\left(t+k\right)}{\partial u\left(t+h\right)} = \sum_{i=0}^{n_d} w_{j,i+1} \begin{cases} \delta\left(k-i,h\right), k-N_u < i \\ \delta\left(N_u,h\right), k-N_u \geq i \end{cases} + \sum_{i=0}^{\min(k,d_d)} w_{j,i+n_d+1} \frac{\partial yn\left(t+k-i\right)}{\partial u\left(t+h\right)} \delta_1\left(k-i-1\right) \tag{36}$$

Note that in the last summation of equation (36) the step function, *δ*, was introduced. This was added to point out that this summation evaluates to zero for *k-i<l,* thus the partial does not need to be calculated for this condition.

6.2 Hessian Element Calculation

The Hessian elements are obtained by once again differentiating equations (34) by *u(t+m),* resulting in equation (37):

$$\frac{\partial^2 yn\left(t+k\right)}{\partial u(t+h)\partial u(t+m)} = \sum_{i=0}^{n_d} w_j \frac{\partial^2 f_j\left(net_j\left(t+k\right)\right)}{\partial u\left(t+h\right)\partial u\left(t+m\right)} \tag{37}$$

where,

$$\frac{\partial^2 f_j\left(net_j\left(t+k\right)\right)}{\partial u\left(t+h\right)\partial u\left(t+m\right)} = \frac{\partial f_j\left(net_j\left(t+k\right)\right)}{\partial net_j\left(t+k\right)} \frac{\partial^2 net_j\left(t+k\right)}{\partial u\left(t+h\right)\partial u\left(t+m\right)} + \frac{\partial^2 f_j\left(net_j\left(t+k\right)\right)}{\partial net_j\left(t+k\right)^2} \frac{\partial net_j\left(t+k\right)}{\partial u\left(t+h\right)} \frac{\partial net_j\left(t+k\right)}{\partial u\left(t+m\right)} \tag{38}$$

The equation (38) is the result of applying the chain rule twice.

7. Simulation Results

The objective of this study is to show how GPC and NGPC implementation can cope with linear as well as nonlinear systems. GPC is applied to the systems with changes in system order. The Neural based GPC is implemented using MATLAB Neural Network Based System Design Toolbox (Norgaard, 2000)

7.1 GPC and NGPC for Linear Systems

The above derived GPC and NGPC algorithm is applied to the different linear models with varying system order, to test its capability. This is done by carrying out simulation in MATLAB 7.0.1 (Mathworks Natic, 2007). Different systems with large dynamic differences are considered for simulation. GPC and NGPC are showing robust performance for these systems. In below figures, for every individual system the systems output with GPC and NGPC is plotted in single figure for comparison purpose. Also the control efforts taken by the both controllers are plotted in consequent figures for every individual figure.

In this simulation, neural network architecture considered is as follows. The inputs to this network consists of two external inputs, *u(t)* and two outputs *y(t-1),* with their corresponding delay nodes, *u(t), u(t-1)* and *y(t-1), y(t-2)*. The network has one hidden layer containing five hidden nodes that uses bi-polar sigmoidal activation output function. There is a single output node which uses a linear output function, of one for scaling the output.

For all the systems Prediction Horicon N_1 =1, N_2 =7 and Control Horizon (N_u) is 2. The weighing factor λ for control signal is kept to 0.3 and δ for reference trajectory is set to 0. The same controller setting is used for all the systems simulation. The following simulation results are obtained showing the plant output when GPC and NGPC are applied. Also the required control action for different systems is shown.

System I: The GPC and NGPC algorithms are applied to a second order system given below.

$$G(s) = \frac{1}{1 + 10s + 40s^2} \tag{39}$$

The Fig.6. Shows the plant output with GPC and NGPC for setpoint tracking. The Fig. 7 shows the control efforts taken by both controllers. The simulation results reveal that performance of NGPC is better than GPC.